(Edited Jan. 21, 2022)

Introduction

In this CAD you will design a 16-bit ALU, which will be used in the datapath of your final microprocessor. This ALU must support two's complement signed arithmetic and the specified instructions in the baseline architecture.

Arithmetic and Logic Unit

The Arithmetic and Logic Unit (ALU) is the main execution unit of your processor. The minimum set of instructions your ALU needs to support consists of:
(i) ADD (ii) AND (iii) OR and (iv) XOR

All of these instructions will be performed on signed numbers. Any operations for unsigned numbers mentioned in the baseline architecture do not need to be implemented. You can implement all of the instructions above with the adder as the basic building block. Additional instructions like SUB (Subtract) and CMP (Compare) could be implemented using the ALU in two cycles. They would be performed by 2's complement addition (A + $\bar{B}$ + 1 (carry_in = 1)).

ADD, SUB, and CMP operations modify the process status registers (PSR). The PSRs are subsequently used by conditional branch instructions. See the appendix at the end and the separate ISA document for details.

There are two general approaches in designing the ALU. The first one is conceptually simpler but the second one may result in a more efficient implementation. Please understand how the two approaches differ.

Separate Logic Blocks

Your ALU must perform four different functions; this could be accomplished as shown in Figure 1, with the adder unit performing ADD, and simple logic blocks performing the other three operations. Then a four-to-one multiplexor (or tri-state buffers) could be used to select the correct function.

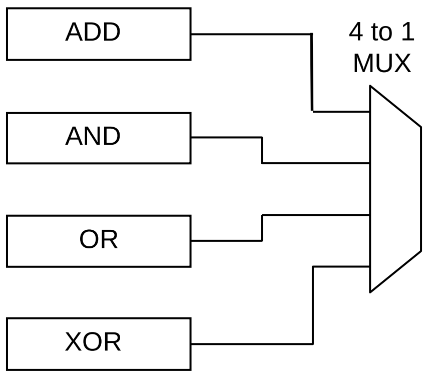

Figure 1: Separate Logic Block Approach

Shared Logic With Adder

An alternative is to share as much of the logic as possible between the 4 operations. If we define propagate and generate functions as:

$$P = A \oplus B \text{ and } G = A \,\&\, B.$$

We can thus generate:

$$A \mid B = P \mid G$$
$$ADD = P \oplus Cin \text{ (Cin is Carry In)}$$
$$Co = G|P \,\&\, Cin \text{ (Co is Carry Out)}$$

You would then use a 2-bit control signal SEL to select the output of one of the four operations.

ALU Design

Carry-Select Adder/ Carry-Bypass Adder

In CAD4, you must implement either a Carry-Select or Carry-Bypass adder based ALU. See Table 1 for the truth table of a 1-bit adder.

Cin	A	B	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 1: Truth Table for 1-bit Adder

PSR Flags

You need to generate PSR Flags (F, Z, N) in CAD4. See the PSR Flags Description provided at the end of this assignment for more information.

Procedure

Schematic Design

The ALU is the key block of your datapath and can be the majority of delay in the execute cycle (if you have a very fast adder, then the memory is likely to be the bottleneck). CAD4 requires you to do a Carry-Select or Carry-Bypass ALU and you need to optimize it for speed through sizing. Remember that ADD, SUB and CMP (immediate versions as well) affect PSR bits. Your ALU should output these flags. The PSR registers will be included in your control unit, so no need to create them now.

Logic Verification

Use NCVerilog to verify each of the four functions. Test the corner cases. Verify your flag generation as well.

Layout

As you design your ALU, think about the overall microprocessor organization. Distribute the buses and control lines the same way you did in the RF. Make sure that these control signals are being driven by a driver. Bring all PSR flag outputs to the boundaries. Since your ALU performs four functions, two select bits should be enough for SEL.

Be sure to run the buses over our cells. Bit-slice width matching with other datapath modules is very important. The final width of your ALU must match the width your RF from CAD3 exactly. Save vertical routing resources for datapath integration. If you find it necessary to use a lot of vertical routing resources within the leaf cell, you could opt to place the ALU on the top or bottom of the datapath such that fewer signals must run over the ALU.

Though you don't need to implement the datapath right now, think about the various sources of inputs to your ALU. They could be from the RF or the program counter (if displacement calculations are done in the ALU) or from the instruction register (immediates). A little thought about the floorplan of your datapath organization in advance will help a lot in getting a good final layout.

Design Verification

DRC & LVS

Analog Simulation

Extract parasitic capacitances. Simulate your ALU's critical path. Provide justification in your README for how you determined the inputs that give the ALU's critical path.

Comments

- NCVerilog traces should show that the 16-bit ALU works for all 4 functions with a few tests per function. Pick tests that cover corner cases. Also show the generation of 3 flags in a separate trace (Make separate .ps files for each function so that bus values are readable in the printout and a separate .ps file showing flag generation). NOTE: Make sure your NCVerilog sims are readable! You will receive 0 points for .ps file where the bus values are unreadable.
- Your ALU must be pitch/bit slice width matched to your RF from CAD3.
- Optimize for area and delay. Your scores for area and delay will be relative to the performance of all groups.
- For this CAD, you are welcome to try any *static* CMOS logic topologies. However, dynamic logic will not be allowed for the reason that dynamic logic cannot be verified using NCVerilog (consider why).

Submission

Please refer to the submission guidelines mentioned in the CAD1 assignment, as they apply to all CAD assignments. Do not lose points over trivial mistakes like incorrect file names, etc!

For CAD4, you need to submit the following:

- 16-bit Carry-Select or Carry-Bypass ALU:
  - Schematic
  - Layout
  - Clean DRC report
  - Clean LVS report
  - PEX (Calibre view)
  - NCVerilog (.ps) files to demonstrate all 4 ALU functions and PSR flag generations (make sure they are readable! – multiple files)
  - NCVerilog (.ps) files with worst case delay added (single file with worst-case ALU function)
  - Simulation .png files showing the longest rise and fall delays of the ALU.
- README:
  - List all the full paths of your designs
  - Briefly explain how you implemented your ALU.
  - Worst case delay numbers and explanations of the delays. BE SPECIFIC. i.e. which paths were critical in determining the delays.
  - Estimate all input pin capacitances (A, B, Cin, SEL…) and explain how you estimated these numbers.
  - Any other comments
- You need to name your directory CAD4 and create a SUBMIT directory in your CAD4 directory. Copy all of your DRC, LVS reports and simulation waveform files to the SUBMIT directory.

Deadline

You must complete CAD4 by **February 10, 2022** by **11pm**.
Do not modify any files in your CAD directory after that time.

Appendix: PSR Flags Description

For the purpose of EECS 427, we define Process Status Register as a 3-bit register storing the following result from ALU operations:

- Carry Over-Flow (F)
- Zero Flag (Z) ie. Operand A == Operand B
- Negative/Less-Than (N) ie. Operand A < Operand B

Take a look in the document "ISA.pdf" uploaded to the Resources/Handouts for details on the flags and instructions. It states 2 other flag registers (C and L) which are set for unsigned numerical operations, but we will not be dealing with unsigned operations in EECS 427 so you can ignore those flags and any instructions which utilize those flags.

Computing Flags

Carry Over-Flow (F)

Adding or subtracting two N-bit numbers can require an N+1 bit number to fully express the result. When the result requires more bits than are available, we have an overflow or "carry condition". Overflow occurs for two's complement numbers under the conditions indicated in Table 2. An easy way to determine overflow is to XOR the carry-in of the high-order bit with the carry-out of the high-order bit. It is left as an exercise for the student to verify that this detects overflow:

$$F = C_{(n,out)} \oplus C_{(n-1,out)}$$

Operation	Operand A	Operand B	Result
A + B	≥ 0	≥ 0	< 0
A + B	< 0	< 0	≥ 0
A – B	≥ 0	< 0	< 0
A – B	< 0	≥ 0	≥ 0

Table 2: Carry Over-Flow (F)

Zero Flag (Z)

Z is set after CMP instruction, which involves subtraction (i.e. Cin = 1). There are two ways to implement this. The obvious way is to implement the zero detection is to OR all the sum bits. The other way is to AND all of the P bits, assuming that P is implemented as an XOR. It is left as an exercise for the students to verify that when (P_1 AND P_2 AND … P_{n-1} AND P_n AND Cin) == 1, the adder output is Zero.

Less Than (N)

N is set after CMP instruction, which involves subtraction (i.e. Cin == 1). This is given by the following expressions, although you should verify that they are correct:

$N = A_{n-1} \oplus B_{n-1} \oplus \overline{C}_{n-out}$

$N = A_{n-1} \oplus \overline{B}_{n-1} \oplus C_{n-out}$

$N = S_{n-1} \oplus C_{n-out} \oplus C_{n-1-out}$

Note that B here is the original value, before it is inverted for subtraction and fed to the ALU. A_{n-1} refers to the sign bit of A. $C_{n\text{-}out}$ refers to the bit that is carried into the addition of the 16th bits (sign bits) of A and B, which is also the carry-out bit of the addition of the 15th bits of A and B.

When implementing the processor in later CADs, be sure to clear the flags when they should be cleared (as well as setting them at the appropriate times). Do not change flags when they should not be changed. This can be implemented by clocking the flip-flop for a particular flag bit whenever an instruction that is allowed to set or clear the bit has been executed. You could include instructions for loading and storing the processor status register (LPR and SPR). These instructions would move data from one of the general-purpose registers into the status register, or read the status register into a general-purpose register. These instructions are not included in the baseline machine; the baseline machine requires the PSR to be modified only by arithmetic instructions and reset. If you are not implementing the LPR and SPR instructions, you can save area by just implementing the flag flip-flops which you need (3 for the baseline processor).